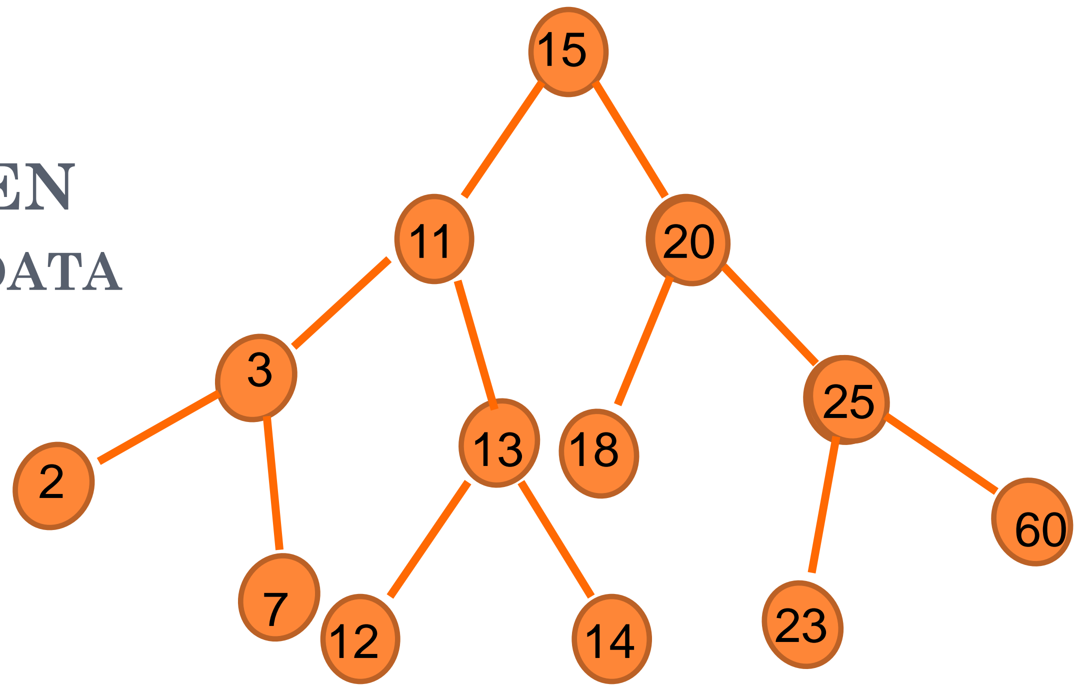
DATASTRUCTUREN VOOR GESORTEERDE DATA



Dr. D.P. Huijsmans

19 sept 2012

Universiteit Leiden, LIACS

Onderwerpen 19 sept

- - Tot nu toe rijen/lijsten met beperkte toegang
- - stack: bovenaan (LIFO)
- - cyclisch array of lijst: vooraan/achteraan (FIFO)
- - linked list implementatie DCLL
- - geheugengebruik beperken: array -> verbonden lijst
- - geordende lijsten: tussenin
- - nadeel sortering bij verbonden lijst: O(n) zoeken
- - binary search voordeel sortering -> BST: O(logn)

Magneetband analogon voor records (vaste of variabele lengte)

- In plaats van een serie bytes kan een magneetband ook gedacht worden als een sequentie van records:
- - van vaste lengte (steeds m bytes/record lezen/schrijven)

- - van variabele lengte (steeds zoveel bytes lezen totdat EOR byte code bereikt is)

- - O(N) toegang (serieel voor/achteruit)

ADT_SingleLinkedRecordsFile RandomAccess Schijfgeheugen

- In plaats van byte voor byte gaat de I/O met vaste of variabele lengte records (EOR) die op het eind een pointer hebben naar het volgende record (NR);
- Laatste record in de rij bevat als next pointer een NR=EOF
- I/O functie die als bij een byte stream file met als uitwisseleenheid het record (i.p.v. de byte) en beperking tot alleen m records voorwaarts kunnen "spoelen"

record	Bloknr nextrec
1	456
2	3
3	2299
4	16
5	EOF

ADT_RandomAccessFile RandomAccessSchijfGeheugen

- Vaste en ook variabele lengte records kunnen sneller benaderd worden bij een opzet waarin naast de afzonderlijke records een array met record-keys wordt bijgehouden; vanuit dit array kan elk record direkt benaderd worden via de pointer die bij de sleutel(key) opgeslagen wordt.
- Array van sleutels kan gesorteerd gehouden worden voor snel opzoeken bepaalde key.

key	1e blok record
3	4050
7	13
12	20134
18	5
76	6789

GESORTEERDE LIJST DOORZOEKEN

- Opgeslagen data elementen oplopend op waarde gesorteerd (geindexeerd)
- Maakt snel zoeken naar aanwezigheid en positie gezochte waarde mogelijk
- Methode binary search (herhaalde tweedeling):
  - Bekijk middelste waarde en vergelijk met gezochte
    - Als gelijk -> klaar return index
    - Als waarde< gezochte bekijk middelste linkerdeel
    - Als waarde > gezochte bekijk middelste rechterdeel
- Randvoorwaarden:
  - Rij even/oneven aantal midden?
  - stopcriterium

Binair Zoeken naar "bepaalde waarde" in gesorteerd array A[1..N]

- BS Algoritme:
- Initieel:
  - Stap=N; index=0; Dir=+1;
- Herhaal zolang Stap>1; anders return(0):
  - Halveren Stap(grootte) en aanpassen Index:
    - Functie half(stap):
      - Stap even-> Stap/2, anders (Stap+1)/2

      Index=Index+Dir*Stap;
  - Test = :
    - Als A[index]="bepaalde waarde" -> return(index)
  - Richting volgende aanpassing index als nog niet gevonden:
    - Dir=-1; als A[Index]<"bepaalde waarde" Dir=+1

Gesorteerd Array zoek voorbeeld

Zoeken naar A[index]="i"

a	b	c	d	e	f	g	h	i	j	k
					6					
								9		

i?	init	ronde1	ronde2
Stap	11	6	3
Index	0	6	9
Test		<	=
Dir	1	1	
Return	0	0	9

Gesorteerd Array zoek voorbeeld

Zoeken naar A[index]="h"

a	b	c	d	e	f	g	h	i	j	k
					6					
								9		
						7				
							8			

h?	init	ronde1	ronde2	ronde3	ronde4
Stap	11	6	3	2	1
Index	0	6	9	7	8
Test		<	>	<	=
Dir	1	1	-1	1	
Return	0	0	0	0	8

Verborgen veronderstelling binary search op gesorteerd array

- Wat gebeurt er als waardes meermalen voorkomen?
- Hoe zou je de datastructuur aanpassen zodat alle voorkomens van een bepaalde waarde gerapporteerd kunnen worden?

Binair zoekalgoritme op gesorteerd array sluitend?

- Zelfs bij veronderstelling dat elke waarde in het array maar 1 maal voorkomt
- Zelf uitzoeken komende 5 minuten:
- Ga na hoe algoritme uitpakt voor gesorteerde array's die maar 1,2 of 3 lang zijn;
- In die gevallen waar het mis loopt, pas het algoritme zo aan dat het in alle gevallen correct werkt.

BINAIR ZOEKEN NAAR “BEPAALDE WAARDE” IN GESORTEERD ARRAY A[1..N]

- BS Algoritme met tekortkomingen:
- Initieel:
  - Stap=N; index=0; Dir=+1;
- Herhaal zolang Stap>1; anders return(0):
  - Halveren Stap(grootte) en aanpassen Index:
    - Functie half(stap):
      - Stap even-> Stap/2, anders (Stap+1)/2

      Index=Index+Dir*Stap;
  - Test = :
    - Als A[index]=“bepaalde waarde” -> return(index)
  - Richting volgende aanpassing index als nog niet gevonden:
    - Dir=-1; als A[Index]<“bepaalde waarde” Dir=+1

Gesorteerd array N=3 voorbeeld

a	b	c
	2	
1		3

?	init	ronde1	ronde2	ronde2	ronde3
Stap	3	2	1	1	stop
Index	0	2	1 of 3	1 of 3	
Test		<>	=	<>	
Dir	1	-+1		-+1	
Return	0	0	a of c	0	

Gesorteerd array n=2 voorbeeld

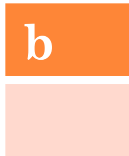
b	c
1	

a?	init	ronde1	ronde2
Stap	2	1	stop
Index	0	1	
Test		<	
Dir	1	-1	
Return	0	0	0

c?	init	ronde1	ronde2
Stap	2	1	stop
Index	0	1	
Test		>	
Dir	1	1	
Return	0	0	0

Element 2 kan niet bezocht worden

Gesorteerd array N=1 voorbeeld



b?	init	ronde1
Stap	1	stop
Index	0	
Test		
Dir	1	
Return	0	0

Element niet gevonden!

Stopcriterium stap>1 faalt bij n=1 of 2

Voor N=1 kan stap initieel minstens 2 zaak oplossen,
Maar voor N=2 blijft 2e element buiten bereik

Vooraf test of A[N] voldoet zou dit oplossen, maar duur gezien
Extra stap

Correct
Binary search algoritme voor elke N

- In “programming pearls” staat een betere aanpak:
- A[N] gesorteerd array N=[1..Max]
- Gezocht index van A[i]=target
- Initieel: low=1,high=Max (**in C++: 0 en Max-1**)
- Zolang low<=high
  - mid=(low+high)/2 (naar beneden afgerond)
  - Als A[mid]=target return(mid)
  - Anders
    - Als A[mid]<target low=mid+1
    - Else high=mid-1

Return(0)

Laat ontdekte bug in binary search algoritme

- Pas laat is nog een mogelijke tekortkoming in dit voorbeeld algoritme ontdekt:
- De regel: mid=(low+high)/2 kan als b.v. mid een 16 bit Integer is en de addressering 32 of 64 bit is mogelijk leiden tot overflow bij low+high;
- Oplossing: een echt voor alle gevallen correct algoritme zou mid als volgt kunnen bepalen:
  - Of mid=low/2+high/2 of nog efficienter
  - Of mid=low + (high-low)/2 (nog efficienter)

Correct binary search algoritme voor alle N [1..Max) voorbeeld

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
							8							
											12			
													14	
												13		

m?	initieel	ronde1	ronde2	ronde3	ronde4
Mid		8	12	14	13
Test		<	<	>	=
Low	1	9	13	13	
High	15	15	15	13	
return	0	0	0	0	13

Binaire zoek boom (knoop structuur)

- Ouder knoop met maximaal 1 voorouder en maximaal 2 kinder knopen
- Als eenzijdig verbonden structuur:

Kind-links pointer	Ouder waarde	Kind-rechts pointer

Verbinding:
edge
link
arc

- Als tweezijdig verbonden structuur:

	Kind-van pointer	
Kind-links pointer	Ouder waarde	Kind-rechts pointer

Ouder waarde:
key:datarecord

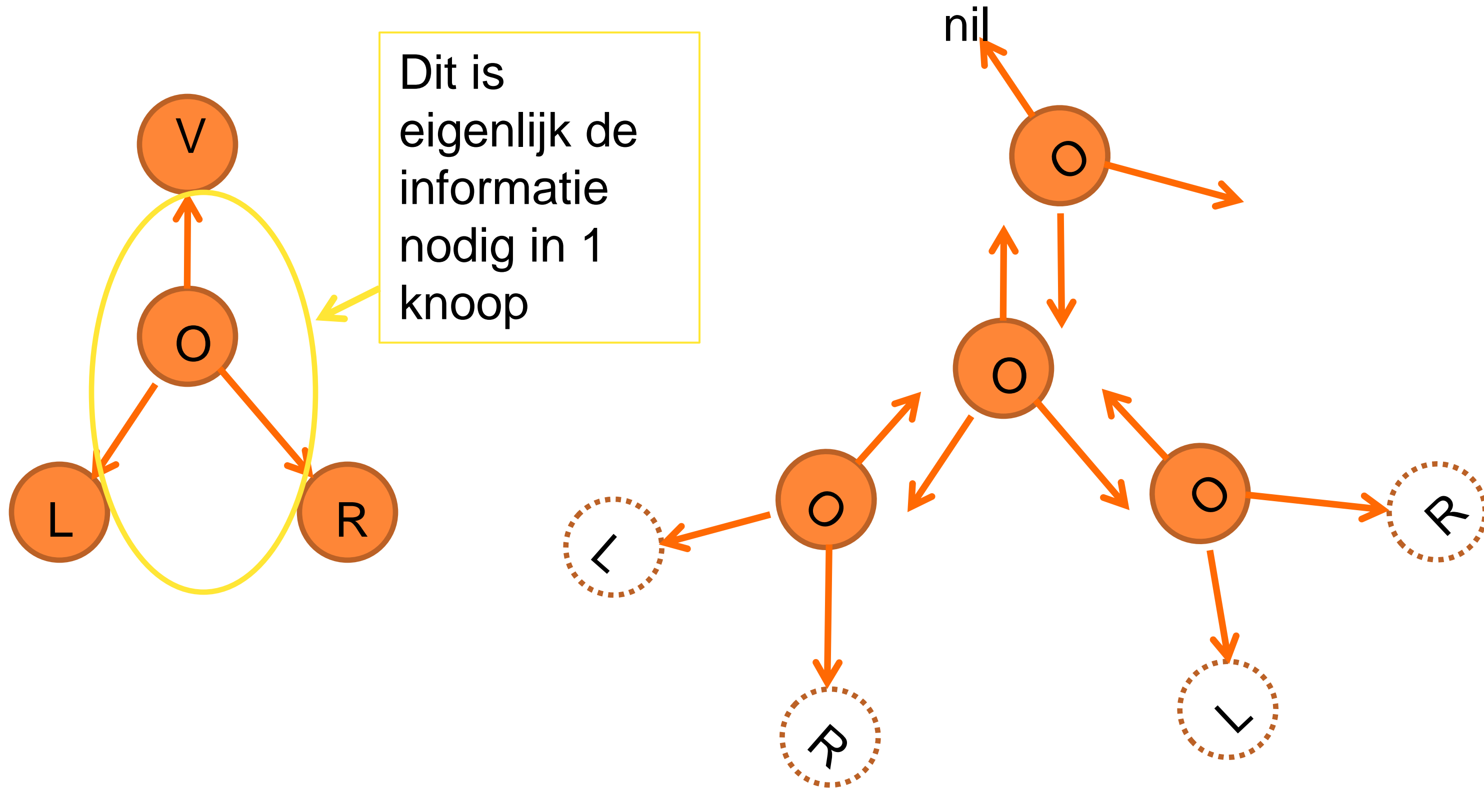
- Leeg knoop element

	nil pointer	
nil pointer	nil waarde	nil pointer

root-element

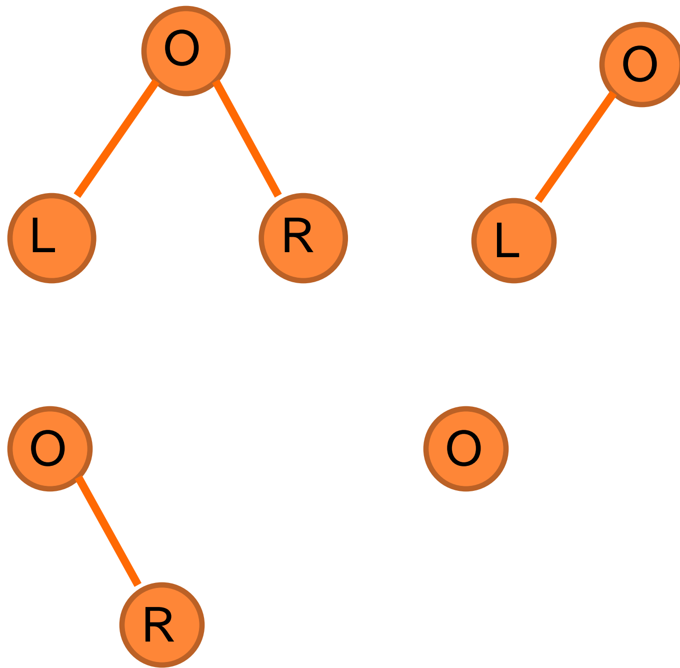
	nil pointer	
Kind-links pointer	key	Kind-rechts pointer

Binary Search Tree ouder knoop met 2 kinder knopen (L&R) en 1 voorouderknoop



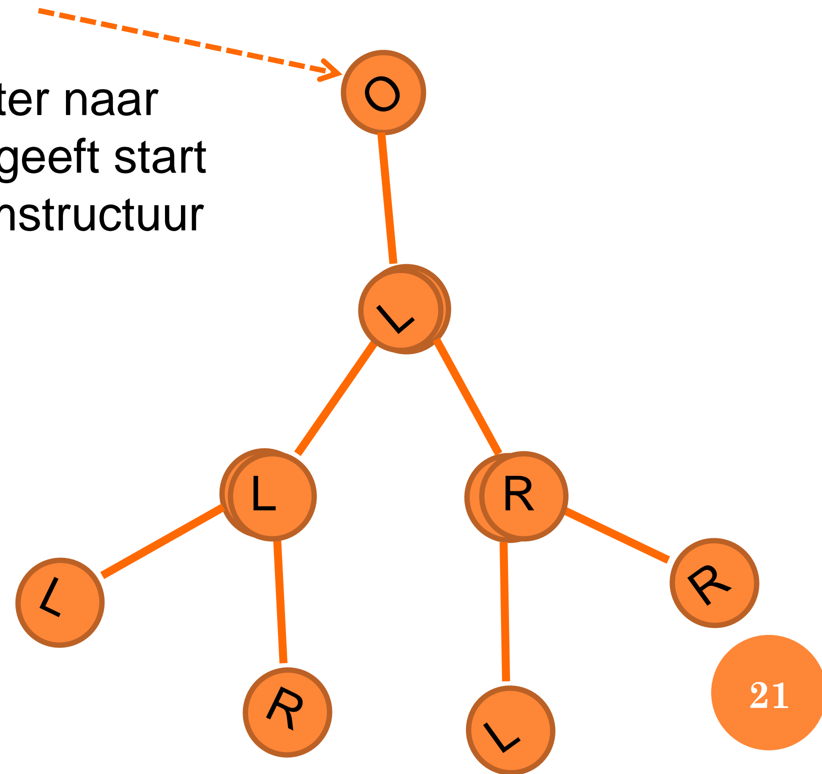
Door samen delen van gerichte pointers kan een ongerichte binaire boom structuur worden opgebouwd die zowel van boven naar beneden, van beneden naar boven doorlopen en langs een willekeurig pad doorlopen kan worden

VEREENVOUDIGDE GRAFISCHE VOORSTELLING KNOOP IN BST



Voorouder pointer niet apart nodig als adres voorouder op stack gezet wordt

Pointer naar root geeft start boomstructuur aan



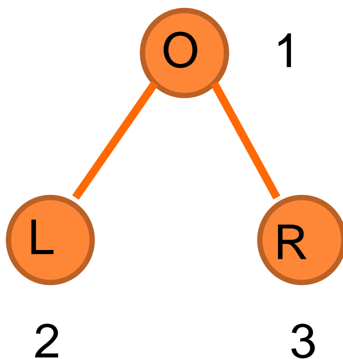
Bouwstenen zonder weergave nil pointers; voorouder pointer en richting pointer

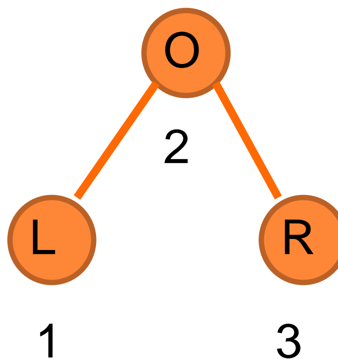
Paden in een boom

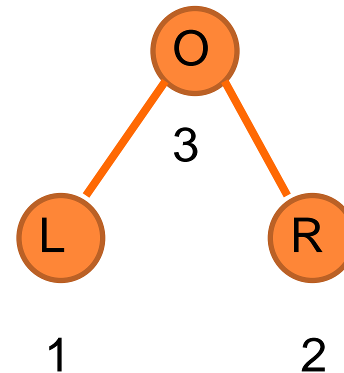
- Knoop zonder voorouder is root (wortel)
- Knoop met als kinderen nil pointers is leaf (blad)
- Geordende Boom:
  - kindknopen in bepaalde volgorde (van L naar R b.v.)
- Simpel pad:
  - verbinding tussen root en leaf waarbij een verbinding maar 1 maal gebruikt wordt
- Lengte van het pad:
  - Aantal knopen langs pad
- Hoogte (height) van een Tree:
  - Aantal knopen in langste simpel pad
- Niveau (level) van een Tree:
  - Knopen op zelfde simpel pad afstand van wortel

Handige afloopvolgordes BST bomen

- Paden waarin elke knoop precies 1 maal opgenomen wordt ook al wordt hij vaker gepasseerd:
- Ouders gaan voor; niveau voor niveau geordend (breadth first):
  - m.b.v. FIFO lijst
- Kinderen gaan voor (depth first) m.b.v. stack:
  - Preorder: OLR
  - Inorder:LOR voor gesorteerde lijst in BST
  - Postorder:LRO





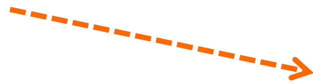


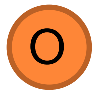
BST
toevoegen knoop met waarde Key

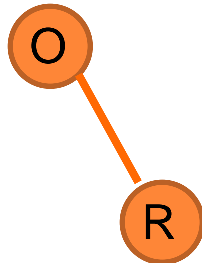
- Mogelijke beginsituaties na zoeken invoegpositie (currentknoop):
- Lege boom: creëer een (root)knoop met waarde key; pointers naar kinderen nil, rootpointer van nil -> (root)knoop.
- Moet in linker subtree van current knoop: creëer nieuwe knoop waarvan key de waarde wordt; linkerpointer currentknoop -> nieuwe knoop; nieuwe knoop met overname van linkerpointer current knoop en nil pointer rechter; currentknoop-> nieuwe knoop
- Moet in rechter subtree van currentknoop: analoog aan linker toevoeging

BST: root of leaf
Links invoegen bij current knoop

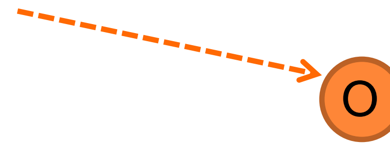
Voor:

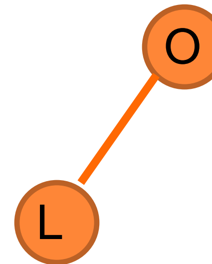


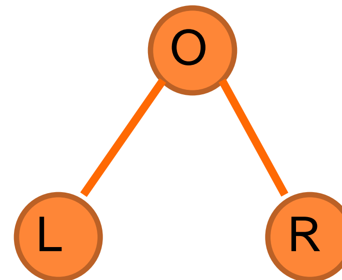




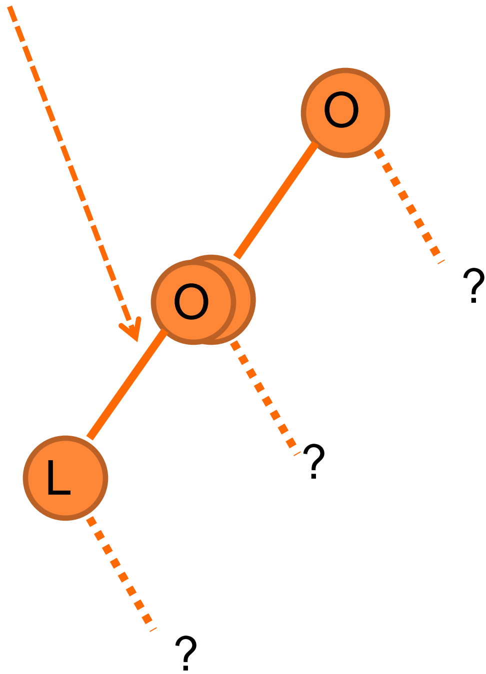
Na:

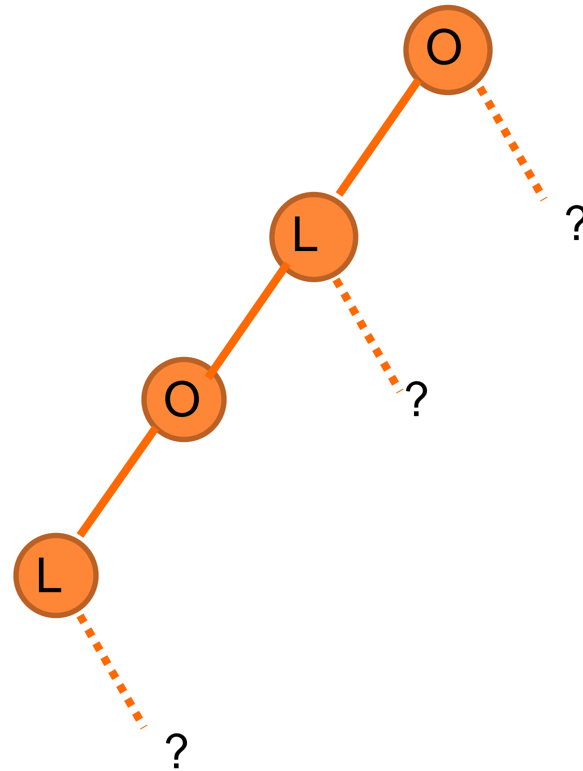






BST: niet root/leaf invoeging links invoegen met lege rechter

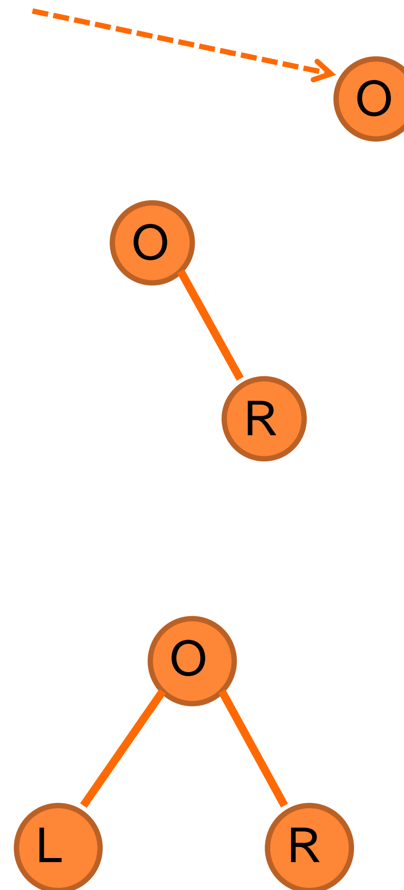




BST:
Rechts invoegen bij current knoop

Voor:

Na:

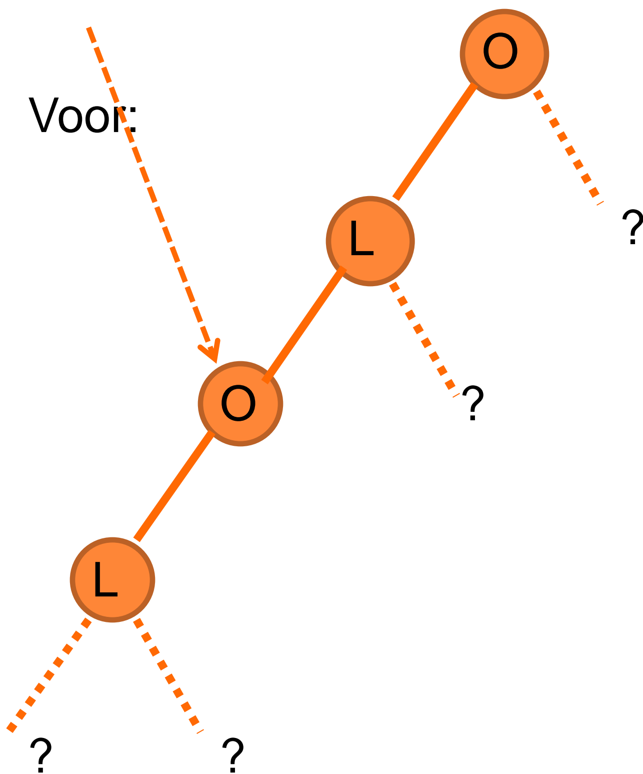


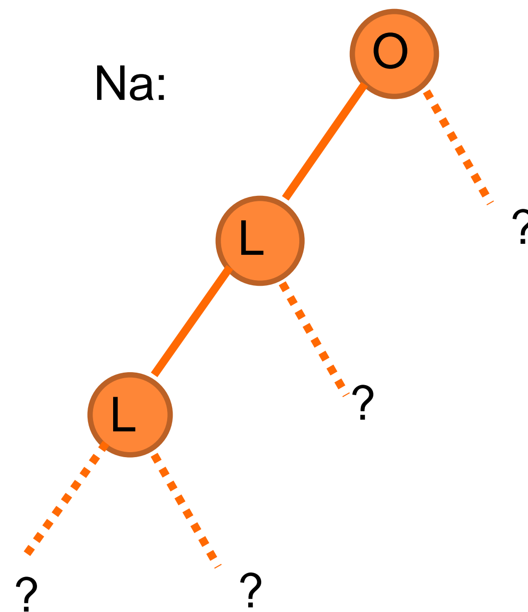
BST:
WEGHALEN WAARDE AAN BEGIN/EIND

- Weglaten van een waarde in een BST zou kunnen gebeuren door de opgeslagen waarde in een knoop op nil te zetten
- Als we ook een knoop weg willen halen uit de BST dan moeten we de volgende gevallen onderscheiden:
- BST is leeg -> niets doen
- BST alleen een root knoop, delete root knoop, zet BST pointer naar nil, size -> 0
- Knoop is leaf (nil pointers als kinderen): knoop daarboven die hier naar toe wees-> nil

BST WEGHALEN WAARDE ONDERWEG

- Delete is alleen makkelijk als de weg te halen waarde in een knoop zit met alleen een linker of een rechter subtree:



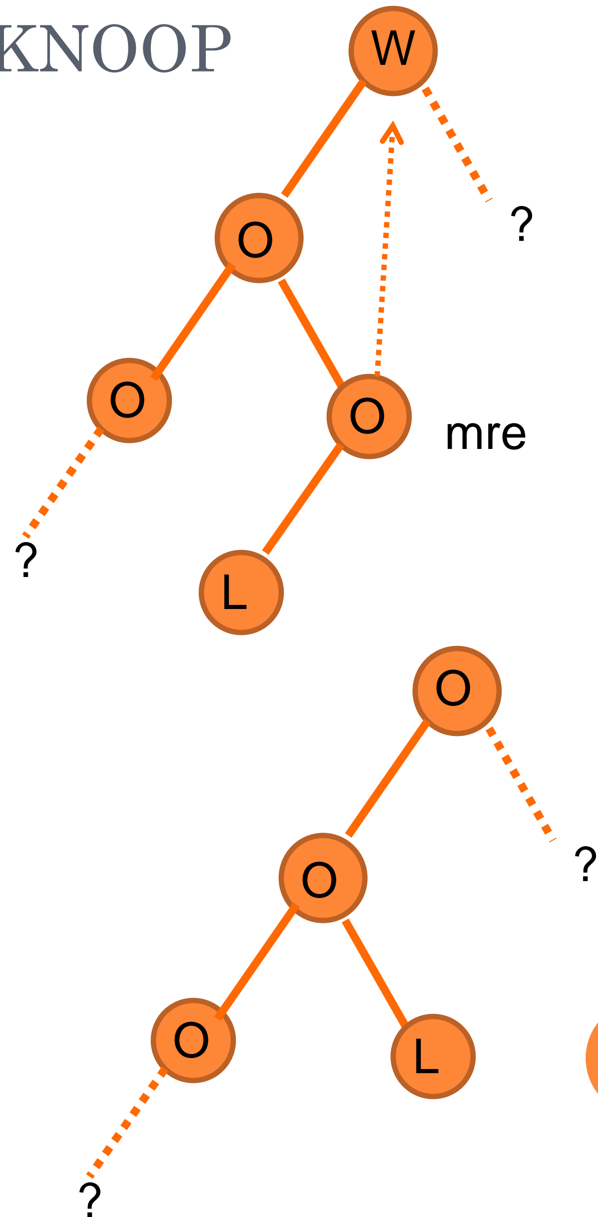


BST: delete knoop met 2 subtrees

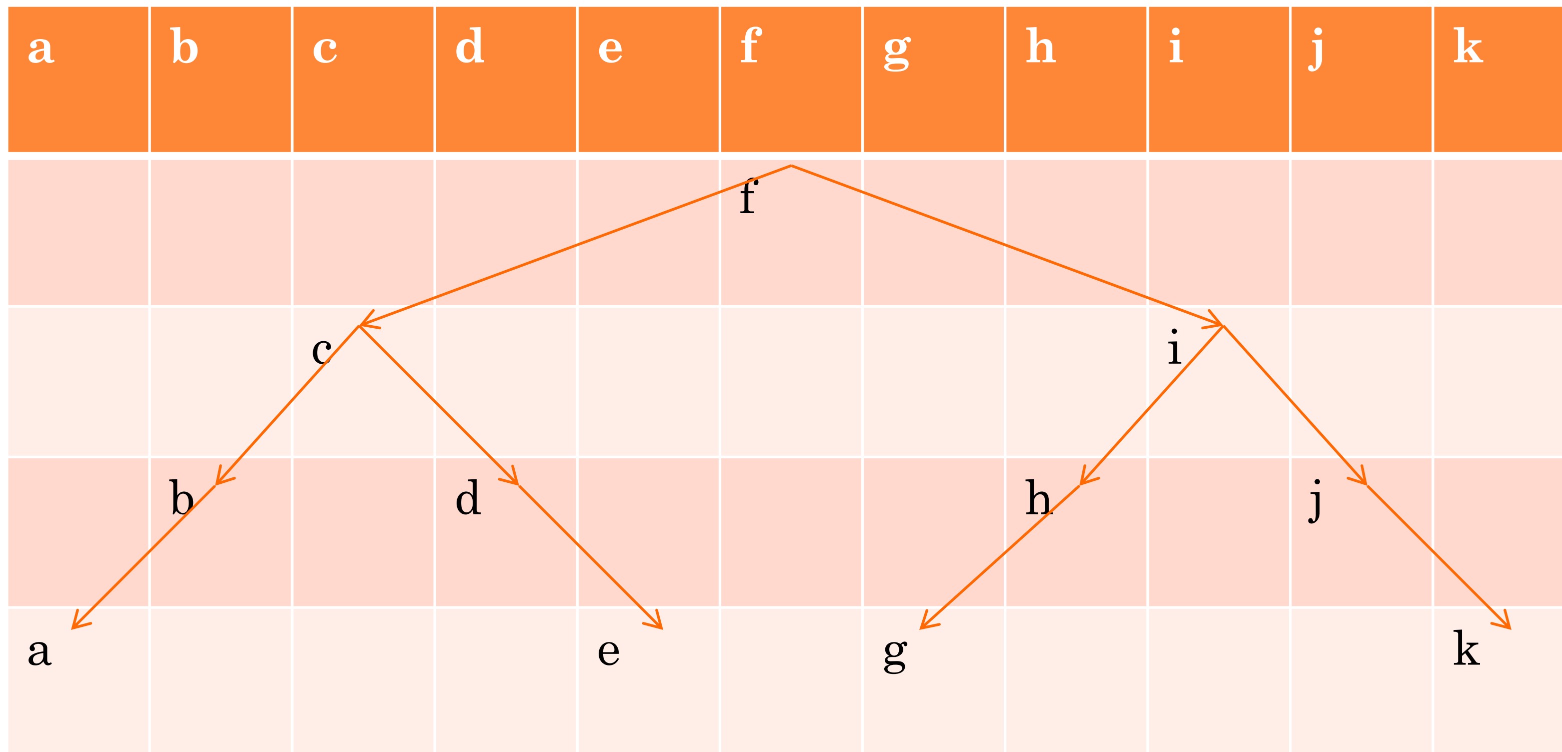
- Als er onder een weg te halen knoop twee subtrees hangen kan maar 1 ervan makkelijk 1 nivo naar boven opschuiven; wat te doen met de elementen in de overblijvende subtree?
- Oplossing1: verhuis waardes binnen boom
- Oplossing 2: loop de resterende subtree af en voeg elk van z'n knoopwaardes 1 voor 1 aan de BST toe
- Oplossing 3: merge de 2 resterende bomen

BST: DELETE ROOT KNOOP

- Net zo lastig en vergelijkbaar is de situatie in een goed gevulde gebalanceerde boom waarin we de root knoop willen weghalen.
- Oplossing: kopieer meest rechtse element in linker subtree naar root en delete de knoop waarin mre oorspronkelijk stond

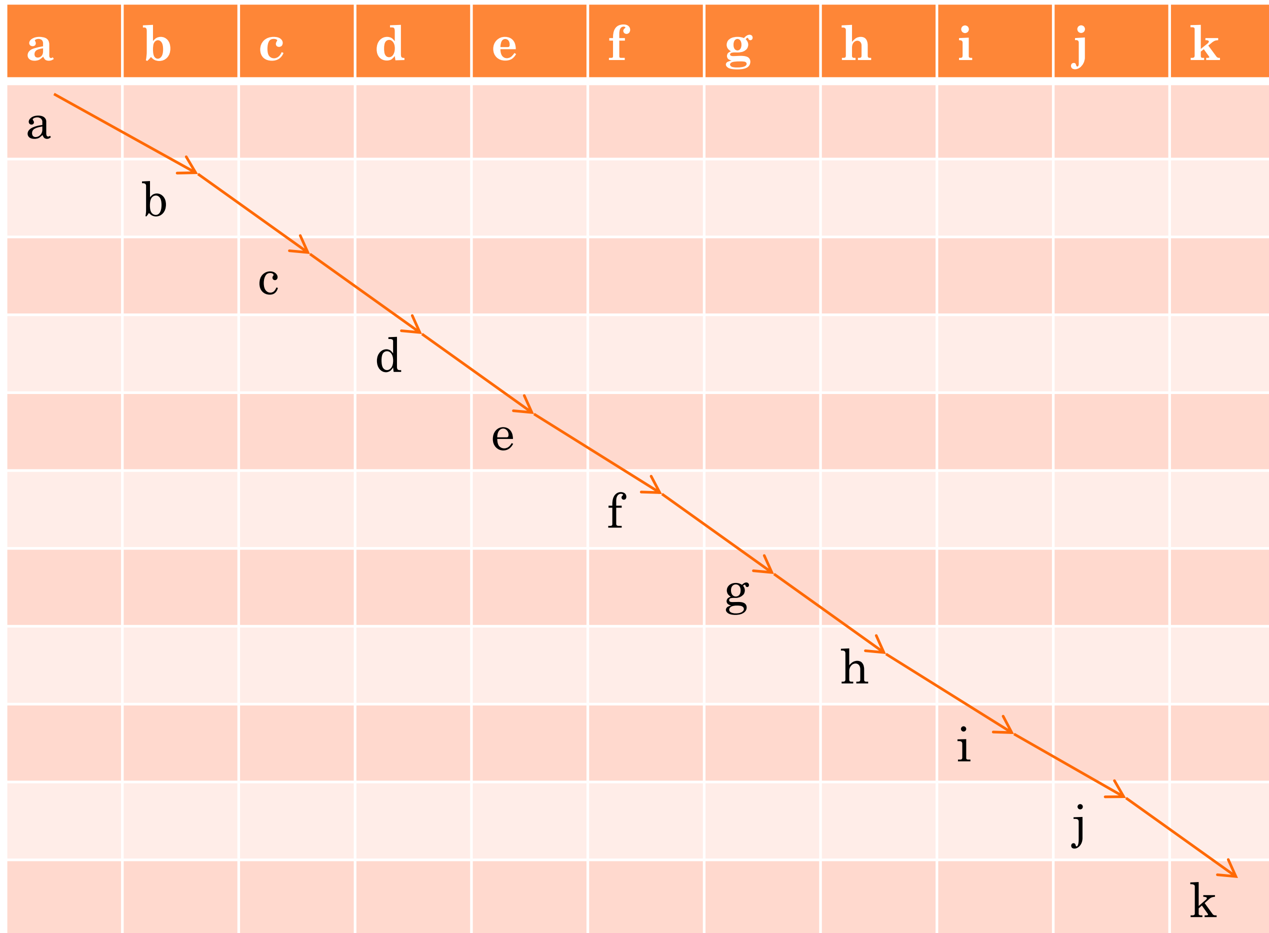


Mogelijke binaire boom opslag voorbeeld



Gebalanceerde BST:
Invoegvolgorde: f,c,i,b,d,h,j,a,e,g,k
Zoeken hierin bijna O(logN)

Ongebalanceerde opslag in BST
vergelijkbaar met gesorteerde enkel verbonden lijst



Dit resultaat heet ook wel “backbone structuur” (komt terug!)

Invoegvolgorde: a,b,c,d,e,f,g,h,i,j,k

Zoeken hierin O(N)

Aantal tweedelingen en aantal knopen in volle BST

- Wortel (root): n=1 knoop (niveau h=1)
- 1 tweedeling: n=3 knopen (niveau h=2): 1+2
- 2 tweedelingen: n=7 knopen (niveau h=3): 1+2+4
- ……..
- h-1 tweedelingen: $n=2^h-1$ knopen (niveau $h=\log_2(n+1)$)

Gesorteerd en gebalanceerd houden

- Zaak is algoritmes voor BST zo te ontwerpen dat:
- Afloopvolgorde en sortering samenvallen
- De boom zo gebalanceerd mogelijk blijft:
  - Bij opbouw van de BST
  - Bij wijzigingen op de BST

BST opbouwen vanuit gesorteerd array

- Een goed gebalanceerde BST kan makkelijk worden opgebouwd als de waardes vooraf bekend zijn
- Zet de waardes in een array
- Sorteer het array
- Voeg elementen aan de BST toe door het gesorteerde array recursief met binair zoekadressering te doorlopen

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
							1							
			2								9			
	3				6				10				13	
4		5		7		8		11		12		14		15